Performance Analysis of Distributed Deep Learning

Sushanta K. Pani, Brendan Hand, Rahul Yalamanchili
CSE 812 - Distributed Systems
Michigan State University

***Abstract*— Machine learning has become something that is used in just about every application of data today. Since it is so computationally intensive to conduct machine learning algorithms on large datasets, it could be beneficial to harness multiple separate computing resources to possibly decrease the computing time. Our work is likely to help data analysts that work with large amounts of data and need a starting point to understand the trade-offs that come with exploring distributed machine learning. We analyzed the training and optimization of deep neural networks parameter over distributed frameworks. This will help us to understand the challenges in deep neural network model learning with large datasets. Furthermore, we uncovered the fact that simply increasing the number of computing nodes and distributing work among them does not directly translate to faster running models.**

I. Introduction

A. Machine Learning and Deep Learning

With the ever increasing abundance of data, there has been incredible growth in the field of machine learning to help make sense of it. Machine learning (ML) gives computers the ability to make predictions based on data, and its applications include face detection and recognition, handwritten digit recognition, and more. Unsupervised learning is a type of ML in which models do not receive ground truth labels with training data, so they need to perform some sort of clustering to group data. This can be useful when preprocessing data, analyzing social networks, and categorizing images. In supervised learning, the models receive labeled training data; this opens up algorithms to be able to classify test data based on the knowledge learned from the training phase. This leads into neural networks that are built using artificial neurons. These artificial neurons take inputs, multiply them by their respective biases, take the sum of these products, and calculate their output using an activation function that resides in them. Neural networks are made up of layers of these artificial neurons connected by weights that the model is designed to optimize to get the best performance. These weights are updated during back propagation which is performed after loss is calculated at the last layer.

Deep neural networks (DNN) are neural networks with a large number of layers which can be divided into three categories: input layer where the data is directly passed in, hidden layers where the neurons with activation functions reside, and output layer where the loss function is utilized. DNNs have become very popular and have attracted the attention of many researchers. It is very useful and applicable in a wide range of research areas such as self-driving cars, voice recognition in mobile phones etc. Interconnected nature of the human brain inspires the core construct of the deep neural network. Just by observing large amounts of data and trained properly, the expressiveness of DNNs provides accurate solutions for problems previously thought to be unsolvable. Due to large amounts of data which motivates learning methodology to consider large numbers of parameters. DNNs often have millions of parameters that require to be tuned during the training phase.

However, the large number of parameters and training data pose challenges in terms of scalability and computation time.[1] Computational intensity and memory demands of deep learning have increased significantly over the years. High performance computing clusters are the present day solution for enhancing performance in deep learning. A trade off between time and accuracy needed to be explore to take a step forward.

B. Distributed Deep Learning

Convolutional neural network is considered to be the default choice for image classification

task.[2] However this requires longer training time for better accuracy. Distributed architecture with parameter servers is considered by many researchers to reduce the training time. Synchronization and communication in distributed architecture are some of the bottle neck in learning the parameter.

Pretrained models such as ResNet-50 which is a CNN trained with ImageNet dataset has initialized parameters before training with new data. This type of model reduces the training time. Distributed architecture along with pretrained models reduces the training duration for larger dataset. This employs the concept of transfer learning which wasn't directly explored in this project.

In this project, we tried to overcome the disadvantages of training with large data for the convolutional neural network and ResNet-50 by utilizing the distributed deep learning methodologies such as Data parallelism and as a stretch goal, Model parallelism.[3][4] We considered some standard datasets like MNIST to design classification algorithms. Unfortunately, we were unable to implement Model parallelism due to the complexity and the time constraint, but we did conduct thorough research on it.

II. Related Work

A. Data Parallelism

Other works have attempted to distribute networks like ResNet-50 or network frameworks like Spark. These approaches focus on distributing via data parallelism and use batch process to improve learning time generally at the cost to accuracy. These papers generally focus on the training of one network or the creation of tools for a framework. Mikami, H. et. al. discusses how to minimize loss and maximize throughput when moving from a small parallel network to a large scale parallel network for the ResNet-50 architecture.[3][5]

Mikami, H. et. al. also proposed the application of a 2d-torus topology for sharing gradient information. This paper utilized sub 64K batch-sizes. This work aimed at improving the performance of massively distributed implementation of ResNet via a synchronous communication between GPUs. [3] SparkNet is a framework designed around Spark and leverages asynchronous communication to attempt to improve performance. [5] ResNet more generally is a Neural Network architecture that aims to lessen the impact of the vanishing gradient problem through the use of residual blocks. These residual blocks use normal layer feed forwarding common in most neural networks combined with skip connections that forward a layers input to the end of the residual block. [6] This provides ways for the network to continue to learn even if some of the layers fall victim to vanishing gradients. However, this provides an extra consideration, should they be adapted to be model parallel, as there will be complex layer boundaries. Park, Jay H., et al. considers the improvement of training efficiency, by optimizing how resources are sent to the networks. This resource awareness attempts to address the communication bottlenecks to decrease training time. [2] These approaches generally work on solving one aspect distributed neural network, as well as providing performance trade off for that one network or aspect they are tweaking. They do not attempt to cross compare distributed networks performances.

B. Model Parallelism

Model parallelism distributes the model architecture layers among different machines, so it is inherently more complicated to implement compared to data parallelism. Each machine is responsible for the computation of part of the whole network. This is especially useful in cases where the model is too big for one node, so this would allow for each node to work with only a segment of the model big enough for it to handle.

When converting a regular deep learning model to a model parallelized version of itself, we need to first assess the RAM capabilities of each available node. The next task of splitting up the model into independent layer sections is one of the most difficult tasks associated with this because things like skip layers and the number of connections across the break need to be taken into account. If there were five edges between layer 5 and layer 6 and there were 25 edges between layer 7 and layer 8, it is clearly advantageous to split the model between layer 5 and layer 6. It is important to take this factor into consideration because lowering the number of edges between the cut would lower the amount of resources spent on communicating

outputs from layer 5 to be fed into layer 6 as inputs. A good approach to find the optimal model splitting would be to use reinforcement learning, but that in itself could add unnecessary complexity. Therefore, finding the optimal splitting of a deep learning model is a difficult task to complete and it is even more so when done manually. [4]

Back propagation is another thing to keep in mind when utilizing model parallelism because the model would have to calculate the loss in the node with the final layer. Then, this loss needs to be used to update the parameters in the reverse order going from the node with the last layer to the node with the first layer. The nodes would have to be able to communicate going forwards as well as backwards to allow for back propagation.

So, the biggest benefit of model parallelism is that no single node is required to load the whole model therefore it would drastically reduce the memory requirements. On the other hand, the downsides include the difficulty of finding the optimal model cutting, and the need to communicate parameters between every few layers depending on how many nodes are being utilized. The overhead required to perform such communication twice over introduces unnecessarily high latency and may cause some nodes to stay waiting for tasks, wasting valuable computing time.

III. Methodology/Technical Approach

A. Overview

The training of DNNs was carried out on three kinds of processors: CPU, GPU, and GPU in different servers. We also tried to compare different Neural networks' performance in each of the different hardware architecture combinations. We utilized data parallelism which means that each data set is divided into many subsets, where the number of subsets are equal or greater than the number of systems under each kind of processor. Here, each system communicated with each other using message passing interface and shared their updated parameters. This update of parameters was continued until it converged to some level. After the convergence, we observed the computation time while training, and accuracy of the model while testing. We used two networks to gauge performance changes each network experiences when moving from local to distributed system. Our paper is something resembling a neural network "shopping guide" to cross compare networks. We implemented data parallelism for ResNet-50 and one additional CNN. After testing, this was revised to implement data parallelism for ResNet-18 and one additional CNN instead. First we implemented single core CPU computation which was performed to make it as a baseline. Then we implemented the GPU implementation followed by multi-GPU. Once the local testing baselines were set, we began implementing the distributed approach of each network. During this stage we collected loss, time, and accuracy performance to get an understanding of how the networks changed when they are parallelized via batches. We proposed a possible stretch goal of implementing model parallelism for both networks. The models tested in this paper will be using the MNIST dataset [7]. This dataset is a collection of images of handwritten digits. It has 60,000 train and 10,000 test examples. Its images have to classify between 10 classes (0-9 digits). The images are normalized to 28x28 resolution.This is a dataset of handwritten numbers that is used often to test Machines Learning algorithms.

B. CNN

In order to encompass the performance metrics found in shallow networks, we created our own shallow neural network. The small architecture consists of two layers of convolution and successive max pooling layers. Every convolution layer was followed by ReLU activation functions to bring non linearity during training. Next, we concatenated feature maps into a single feature vector 800x1. We passed these feature vectors through two fully connected layers; the first of which had a hidden size of 500 and was followed by the ReLU activation function, and the second of which had hidden size equal to the number of classes. We applied Softmax to the output of this dense layer and used Softmax negative log likelihood as our loss function. The ground truth labels were represented as 10-dimension one-hot vectors where each dimension is a separate digit category. We have 10 classes for the 0 to 9 handwritten digits. We used batches of size 32, 64, 128, 256, 512, 1024, 2024,

4096, 8192. We tried experimenting with different learning rates but 0.01 is our optimal learning rate to have a trade-off with classification accuracy. We used the stochastic gradient descent optimizer, and implemented this network in PyTorch. We ran this experiment on 1 CPU and a combination of GPUs (1, 2, 4, 8) to gauge our model performance. We utilized the test set that we described in the data section for evaluation purposes.

C. ResNet

Since ResNet is a deep network that can utilize pretrained weights, our performance expectations vary from that which can be achieved be the traditional CNN. ResNet traditionally uses a 224x224x3 input layer to a convolutional layer and the particular library used a 7x7 kernel. To compensate for MNIST's smaller grayscale image the input channels were reduced from 3 to 1 and the kernel size was reduced to 3x3. The training parameters were the same as the shallow CNN. The expected performance of the network would be greater than a similarly deep neural network. This is due to ResNet utilizing residual connections to help minimize the chance for the vanishing gradient problem to hurt accuracy. Regardless, the depth of the network was expected to produce some problems when training. The ResNet-18 Model used in the results did not utilize transfer learning to start its weights based on the ImageNet dataset. This decision was made as the performance experienced in preliminary tests was not improving outside of the initial epoch. It is likely that with larger batch sizes this may have aided performance, but would not have prevented the overfitting.

D. Performance Metrics

1) Loss: In the last layer of each CNN, the loss function during the training phase is used to measure how much the model parameters need to be updated by during back propagation. The output of the loss function is not bounded, and it is generally a negative value. Our tests were conducted using the negative log-likelihood function.
2) Accuracy: This value determines what percent of predictions on the test set are correct. We compared the final accuracy of each model when assessing them.
3) Time: We measured the amount of time it took to train each model in milliseconds or seconds. This is also referred to as training time or runtime.

IV. Design Rationale

In order to complete the goal of producing a buyer's guide to distributed neural networks we need to test our results both in a local and in a distributed manner. Likewise, even if our ability to test large distributed systems were limited, we need results from the smaller tests in order to project how they may perform on larger physical hardware tests along side larger parameter space tests. This paper or any paper on this subject cannot exhaustively search the parameter space and hardware setups due to time and cost constraints. Nonetheless, this paper will try to give a fair estimate to the parameter space and hardware analyzed. Cluster nodes with identically computing devices do not have to consider some of the constraints on impromptu networks and local computation does not have to account for the same model synchronization found in computing clusters. Together, we can propose how these networks may perform in a hypothetical local impromptu network and what considerations need to be made to ensure their optimal performance. The most important metric we were looking at and the central element of our design rational was the effect of batch size.

V. Technical Results and Analysis

A. Experiment I: CNN

In our first experiment, we analysed the performance of our model architecture for data parallelism approach with our chosen performance metrics discussed in the previous section. In terms of the specific device specifications for this experiment, the CPU was an Intel(R) Xeon(R) Silver 4214 CPU @ 2.20GHz and all of the 8 GPUs were identical NVIDIA GeForce RTX 2080 Ti. We found that the average loss remained almost the same for different sets of CPU and GPU combinations. Overall, with the increase in batch size, the average loss increased across the board. The classification error also remains similar for

different combinations of set of CPU and GPU combination; however, it decreases with any increases in batch size.

To our surprise, the training time observed is quite different from our assumption that it should decrease with increase in computing units. The single GPU trains faster as compared to any other combination of GPUs. Also the increasing order of training time are two, four, and eight GPUs. Therefore, in terms of training time (runtime) the model trained with one GPU is found to be the best.

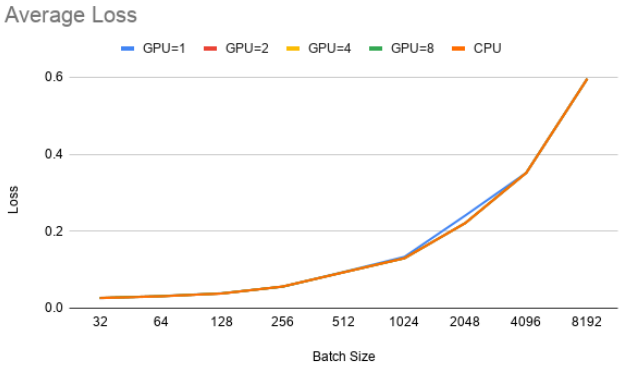

Fig. 1. Negative Log-Likelihood Loss of the shallow CNN over the test data

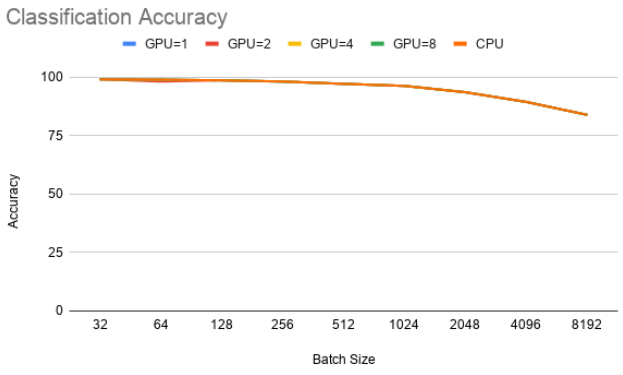

Fig. 2. Accuracy of CNN classification on Test Data (max 100)

B. *Experiment II: ResNet-18*

The results for ResNet were conducted by testing ResNet-18. This was a change from the original plan of testing ResNet-50. This change was made as the deeper network, ResNet-50, was not easily trained on the weaker hardware used in some of the tests. The graphs show that with the negative log-likelihood loss function and low batch sizes ResNet-18 was prone to large amounts of loss and over fitting. The loss in some cases was so low it maxed out the storage container yielding a nan or not a number. This loss was made less negative when the batch size increased. This increased batch size tended to not only improve the loss but the testing accuracy as well. The Figures 4,5,6 were produced from results of tests run on a server cluster. The performance varied little with the change in hardware. This is most notable among the Training time where the time to train varied little from 1 to 8 GPU(s).

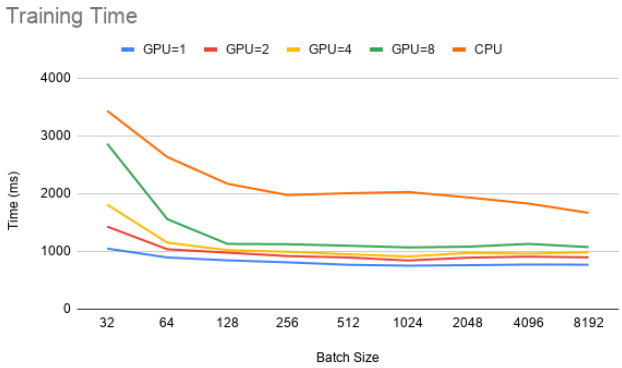

Fig. 3. The Training time of the CNN in milliseconds

Given deeper networks higher promise for distributed approaches. Some research was conducted to determine whether small networks consisting of only a handful of nodes could improve performance. Limitations via hardware prevented the practical tests of this, but the nodes were tested to analyze the prospect of this approach. Figures 7,8, and 9 show the different performance on different levels of hardware. The Intel CPU was a laptop CPU from about 5 years ago while the AMD and GTX results were from the current year, 2020. There was a large increase in run time from the old hardware to the newer hardware. There was an odd doubling in runtime for the 512 batch test. This was present on two runs, but a third run was conducted with more values around that batch size. This showed a lower spike in run times, but it still was a local maxima in training time. This can be found in figure 10.

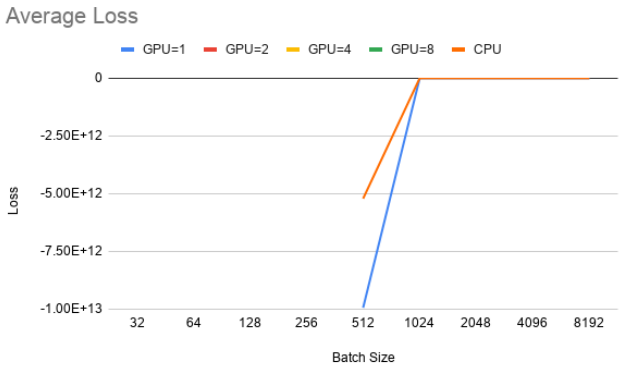

Fig. 4. Negative Log-Likelihood Loss of the ResNet-18 model over the test data

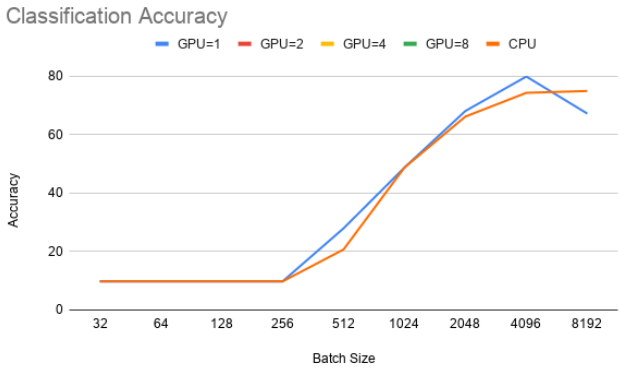

Fig. 5. Accuracy of ResNet-18 classification on Test Data (max 100)

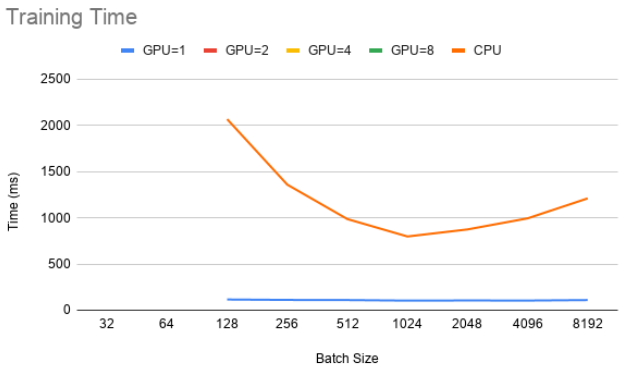

Fig. 6. The Training time of the ResNet-18 in seconds

VI. Future Work

The project [3] achieved its goal of exploring the feasibility of a buyers guide to neural networks, but the topic, at large, still has some unanswered questions. The first open question is whether or not multiple neural networks share the same performance curves with respect to batch size. Reason would seem to indicate that similar

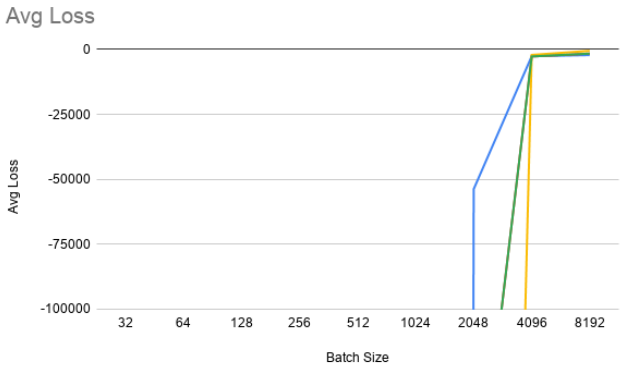

Fig. 7. The Test Loss of ResNet-18 on Local Hardware. Blue GPU GTX 1660Ti, Yellow CPU AMD Ryzen 5 3600, Green CPU Intel Core i7-5500U 2.4 GHZ

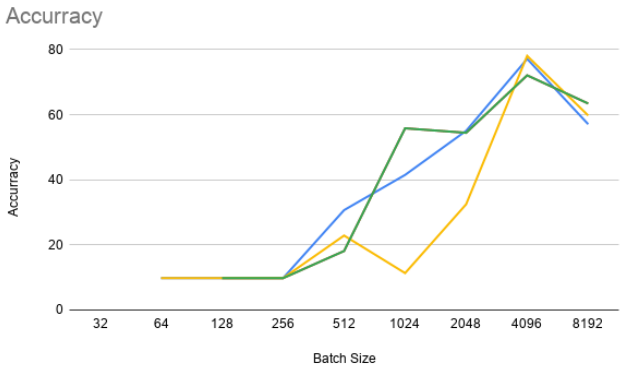

Fig. 8. The Test Accuracy of ResNet-18 on Local Hardware. Blue GPU GTX 1660Ti, Yellow CPU AMD Ryzen 5 3600, Green CPU Intel Core i7-5500U 2.4 GHZ

Fig. 9. The Training Time of ResNet-18 on Local Hardware. Blue GPU GTX 1660Ti, Yellow CPU AMD Ryzen 5 3600, Green CPU Intel Core i7-5500U 2.4 GHZ

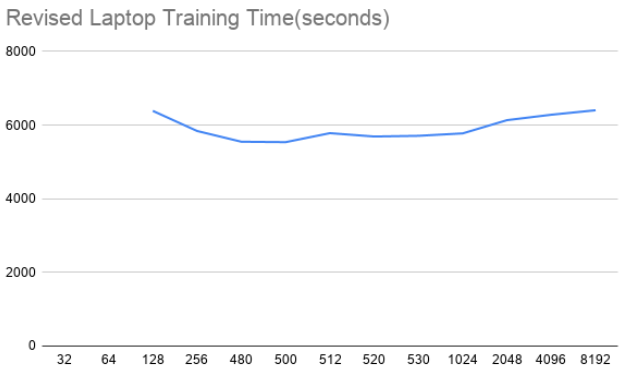

Fig. 10. The Revised Training Time of ResNet-18 on Local Hardware. Blue GPU GTX 1660Ti, Yellow CPU AMD Ryzen 5 3600, Green CPU Intel Core i7-5500U 2.4 GHZ

networks would have similar curves accounting for and shifting or scaling. This likely is not the case for all networks as different network depths, widths, and special features change how easily they train on small batch sizes and how likely they are to experience the vanishing gradient problem.

This paper also highlights the large generational gaps in performance from devices that are different ages and different levels of ml hardware optimization. The fact that RTX graphics cards have a performance near 20x that of GTX graphics of the same model year provides an interesting challenge when designing the distributed network paradigm. The typical research for distributed neural networks has mostly focused on identical nodes or at least clusters with similarly powerful nodes. This model only needs to account for min and max batch sizes based on the dynamic load of the cards assuming they are shared with other processes. Academic environments could eliminate this performance as a means of testing their given topology scheme or the like, but it comes at the cost of representing the true performance of the distributed network in a business setting. The proposal we have is to assess the feasibility of dynamic batch sizes based not only on dynamic load constraints but the static constraints of relative node performance. This would be necessary as even in an ideal case where communication delays to not occur we can analyze the speed of the cluster of two nodes. This cluster would have one is 20 times faster than the other. In this ideal case, it would stand to reason that the weaker node could at most take 1/21 of the overall load of the task. The two things to test with this potential future work is the real optimal ratio of data from one node to the other and what effect this has on the overall performance. We predict that it would be not improve either significantly if the necessary lower batch size for the weaker node is outside of the trainable batch size range.

VII. Workload Distribution

A. *Sushanta*

- Data-set loader
- CNN Model design
- Computational setup and remote utilization

B. *Brendan*

- ResNet modifications
- Test ResNet and result analysis
- Ad Hoc network research

C. *Rahul*

- Initial research
- Model validation (fine tuning parameters)
- Model testing

VIII. Summary and Lessons Learned

We discovered that training deep networks in a distributed manner is not always a good choice. Distributed computing adds overhead from parameter synchronization, and the ratio of coordination to computation determines the performance impact of distributed computing. Distributed methods work best when computation advantage outweighs coordination setback. In the case of medium or large network and datasets, this ratio is lower as the computation per iteration is quite high. Hence, in such cases a distributed training approach is useful. However, in the case of small architecture and dataset, the ratio of data transfer to computation is high as less computation is performed per iteration. The overhead due to synchronization and transfer of data and parameter outweighs the computational advantage due to the addition of machines or processes. Moreover, the setup time and hyper-parameters sharing are also added in overhead. These considerations also apply to ad-hoc or impromptu clusters that use would use

multiple machines to train one model. The understanding of how disparate the computational power of different machines provides a weak basis for some of these networks to be viable. Furthermore, the frameworks and communication protocols themselves provided a real world complexities like choosing specific graphics drivers or compiling binaries that would make these non-consumer friendly approaches.

REFERENCES

[1] Tal Ben-Nun and Torsten Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Computing Surveys*, 52, 02 2018.

[2] Jay Park, Sunghwan Kim, Jinwon Lee, Myeongjae Jeon, and Sam Noh. Accelerated training for cnn distributed deep learning through automatic resource-aware layer placement. *arXiv Preprint*, 01 2019.

[3] Hiroaki Mikami, Hisahiro Suganuma, Pongsakorn U-chupala, Yoshiki Tanaka, and Yuichi Kageyama. Imagenet/resnet-50 training in 224 seconds. *ArXiv*, 11 2018.

[4] Ruben Mayer and Hans-Arno Jacobsen. Scalable deep learning on distributed infrastructures: Challenges, techniques and tools. *ACM Computing Surveys*, 03 2019.

[5] Philipp Moritz, Robert Nishihara, Ion Stoica, and Michael Jordan. Sparknet: Training deep networks in spark. *CoRR*, 11 2015.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 06 2016.

[7] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010.